CMSC 421
Final Project Design

[BRYAN SOLIS]
[MAY 12, 2019]

1. Introduction
   1.1.System Description

During this project, we are assigned to add two new features to the kernel. We had to create a firewall and some type of protection for our files like an access control. The two parts are separate, and we could have started on either because they do not have a connection in any type. The firewall implementation will have to block, unblock some ports, give some count about how many times a port was tried to access as well as a way to reset the whole data structure. The firewall should not close previous connections once it gets a new command to block a used port. The file system implementation will have almost the same characteristics. The implementation will be able to block and unblock the files. Also, the files should have a count of how many times the user tried to access a blocked file. The tricky part is that even root should not be able to modify these files if they were blocked.

   1.2.Kernel Modifications

In order to fulfill the requirements of the project, I used and modified the following kernel functions/files:

Functions Modifed for firewall

- /net/socket.c:
    - int __sys_connect(int fd, struct sockaddr __user *uservaddr, int addrlen) On this function, I added one function to check that if there exit outgoing TCP requests and it will block it if certain ports is on my data structure list.
    - int __sys_bind(int fd, struct sockaddr __user *umyaddr, int addrlen) On this function, I added a function to check that the socket is trying to bind to a socket and it blocks both TCP or UDP connections.
    - int __sys_sendto(int fd, void __user *buff, size_t len, unsigned int flags, struct sockaddr __user *addr, int addr_len) On this function, I added a function to check outgoing UDP request and it will block it if certain ports is on my data structure list.

Function created for firewall

- /fireWall/firewall.c and firewall.h
    - long fw421_reset(void) — this function will delete any data store in our data structure. This function can't be used by user, its only at root level.

- long fw421_block_port(int proto, int dir, unsigned short port); —this function will create a new “blockPort” or instance for the port we want to block and it will add it to our data structure.
- long fw421_unblock_port(int proto, int dir, unsigned short port); — this function will be responsible for iterating our list of blocked ports and if there is a port the user want to unlock, and it removes the port from the data structure.
- long fw421_query(int proto, int dir, unsigned short port); — this function will be responsible for iterating the list of blocked ports and return a specific number/count that a certain port was accessed.
- int checkTCP_UDP (int proto) -this function will check if given int proto is either TCP or UDP
- int searchPort(int proto, int dirPort, struct sockaddr_storage *addrcheck)- this function will be dealing in finding the address of the socket and search it on my data structure.
- int checkPort(int proto, int dir, unsigned short port)- this function will check if there exists a blocked port in our data structure
- int blockWrite(void)-this function will be the mutex for locking a write mutex.
- int unblockWrite(void)- this function will be the mutex for unlocking the write mutex
- int blockRead(void)- this function will be the mutex for locking the read mutex
- int unblockRead(void)- this function will be the mutex for unlocking the read mutex

Functions Modifed for files

- /fs/namei.c:
  - int link_path_walk(const char *name, struct nameidata *nd) – this function will be the one I will modify in order to check if a file is block, and if it is blocked send an error correspondingly.

Functions created for files

- /fireWall/acFile.c and acFile.h
  - long fc421_reset(void); — this function will delete any data stored about blocked files in our data structure. This function can’t be used by user, its only at root level.
  - long fc421_block_file(const char *filename); — this function will create a new “blockFile” / instance for the path file we want to block and it will add it to our data structure.
  - long fc421_unblock_file(const char *filename); — this function will be responsible for iterating our list of blocked files and if there is a file the user want to unlock, and it removes the file from the data structure.

- long fc421_query(const char *filename); — this function will be responsible for iterating the list of blocked files and return a specific number/count that a certain file was accessed.
- int searchFileID(const char __user *fileName, unsigned long* pathId)—this function will be responsible to find the kernel path of file and the ID of certain file.
- int checkFileID(const struct path filePath) – this function will be responsible to take the struct path from link_path_walk so we can get the path and block it in our data structure.
- int blockWriteF(void) – this function will be responsible to lock the write mutex of the data structure.
- int unblockWriteF(void) – this function will be responsible to unlock the write mutex of the data structure.
- int blockReadF(void) – this function will be responsible to lock the read mutex for the data structure.
- int unblockReadF(void) – this function will be responsible to unlock the read mutex for the data structure.

2. Design Considerations

2.1.Network Firewall

For the network firewall, I first started creating a linked list for the main data structure. I would later implement a red black tree to get faster performance. Sadly, I had problems compiling with the red black tree, so I jump back to the linked list to finish the project. Later, I decided to check the files that are in charge for dealing TCP and UDP connections. In class, the professor gave hints about which files might be able to check this and by checking this function I notice that the function connect(), bind(), and sendto() are the function that I used to block the ports because these function deals with connection to prevent a connection either TCP or UDP that has been blocked on our list.

I would also create helper function ”searchPort” to deal with searching the ports in our data structure in case we need to get a faster search and we do not have to go to the general syscalls function to check if a port is blocked or not. The same way with the checking TCP or UDP. This will prevent unnecessary ways to go to the main function to check for ports or type of connections.

In order to prevent data alterations, I have created a simple way to create a mutex for read and another mutex for write. The read mutex will be dealing when we will only read data from our data structure to make sure the data doesn’t get changed during its critical section. The write mutex will be dealing with locking while we are either writing data on our data structure during its critical section. I was planning to use semaphores in the future but due to the lack of time, I could not implement those in time. I would consider this for future improvements.

2.2.File Access Control

For the File Access Control, I have created the same data structure as the firewall. I used a linked list with a potential change to a red black tree. I have used linked list at the beginning because I learned how to use it for project one so I did not want to jump to another data structure that I do not how to modify or create. I wanted to use my time wisely and make sure something works before I could jump to another data structure. Also, I have taken advantage of the features of the inodes in the kernel that deals with the files and its paths. The inode has a special ID for each of the files and each file has a different ID so I decided to use this as a way to store this ID in my data structure. This ID will help me find files or block files path at the same time because every directory or path are considered a file for linux and each of them has a special and unique ID. Later, I have checked potential function where I can modify in the kernel in order to prevent that files are written or read. The functions link_path_walk() is the one I will be modifying in order to prevent that a file is written in a locked path or send error if a blocked file is tried to access. The files mentioned about mostly deal with the path of creating a file in a blocked path. Sadly, I couldn't make the absolute path to be block. Since the path is being built iteratively in the for loop of the link_path_walk(). I tried many parts inside the for-loop in order to find where the absolute path ends. I have put comments on the function link_path_walk() so they can see where I tested my function.

In order to prevent data alteration, I have used the same type of mutex that I used for the first part of the project. I simple way to use mutex. A read mutex and a write mutex. These two mutex will be responsible to handle the locking part in our data structure to make sure that once we are reading data from our data structure and it's on its critical section it can be lock and its safe state. The same method is used for write mutex.I

3. System Design

3.1.Network Firewall

The network firewall structure has a struct called blockPort. I have decided to add 4 variables inside this struct because while reading some of the syscall function we are to implement, it deals with int proto, int dir, unsigned short port. Therefore, I used the same type of variables inside my struct "blockPort" and I called them almost the same so that I do not get confused later on the implementation. The names are typeProto which will deal with the type of protocol we will getting, dirPort which will deal with the direction which is either 0 for outgoing or 1 for incoming, then the variable portToblock will be the one that deals with information for port. The port that we will save to block. Lastly, another variable called count. This variable will keep track of how many times a certain port was tried to access. Next, the function int checkTCP_UDP, this is a helper function that will help me figure out if int proto is either TCP or UDP. This function will return the IPPROTO_TCP or IPPROTO_UDP accordingly. IPPROTO_TCP is chosen for the protocol if the type was set to SOCK_STREAM and IPPROTO_UDP if the protocol is to be set SOCK_DGRAM. This is certainly a great helper function because I

won't be dealing with problems of TCP or UDP connections. Next, the function called int searchPort(int proto, int dirPort, struct sockaddr_storage *addrcheck) will be dealing with a lot of things such as checking that its UDP or TCP with the helper function mentioned above. Checking that the given address is in the right form of little endian using the building function be16_to_cpup which will do the transformation for us. Also, I will be checking that if the data structure "listBlockPort" , where I have all the list of blocked port is , is empty. If the list is empty, it will return 0 and exit because it's nothing to do on an empty list. If it is not empty and it's a match on the port, proto, and dir we will return an error saying a block port was tried to access and we added a count to the current blocked port. Also, the mutex were used during the reading and writing to our data structure correspondingly. Next, the int checkPort() will be a helper function that it will check if there is a math for a given port, dir and proto. If its return -EEXIST. If not return 0.

The syscall function long fw421_reset(void), this function will be responsible to clear all the blocked port on my data structure. It will check this has root level, and it will iterate through the list of "allBlockPorts" and it will erase them one by one. It is important to note that the write mutex will be present during this action.

The next function long fw421_block_port(int proto, int dir, unsigned short port) will be responsible for adding a new port that we want to block on our data structure. It will check that proto is either TCP or UDP, if it is not then return -EINVAL. This is just a friendly case to make sure than nothing else is accepted except for TCP or UDP protocols. Another friendly action will be checking if int dir is 0 or 1, since on the description we will be only dealing with this. If its not 0 or 1 then return -EINVAL. Later, we will kmalloc the new blockPort struct we want to add to our list and then we add it. The write mutex will be present during this action.

The next function long fc421_unblock_file(const char *filename) will be responsible to remove any port desired port from our linked list. It will follow the same structure as fw421_block_port. It will check for invalid protocol and dir. It will iterate on our list and if there is a match in the int proto, int dir and port, the match blockPort will be deleted from the linked list and return 0. If there was not match found, it will return an error with –ENOENT.

The function modified on the kernel are connect(), bind() and sendto(). I used the same function searchPort on all 3 function. I created this function to check that a certain port is block on my linked list. At the beginning, I used different function to block on connect(), another function to block on bind() and sendto(). I know that connect()e for TCP request and sendTo() for UDP request. I wanted to do the checking at the same time that's why I used the helper function int checkTCP_UDP to make sure im checking both. I also using the function seachPort on the bind() function in order to prevent connection for either TCP or UDP. If a match of a port is fond then an error of -EINVAL is return.

3.2.File Access Control

The file Access Control structure has a struct called blockFile. I have decided to add 2 variables inside this struct because while reading some of the syscall function we meed to implement and deals with char* filename. Therefore, I used some variables inside my struct “blockFile” called int count and unsigned long fileID. The variable count will have information about a file and how many times it was tried to access. The variable fileID will have the ID number of a certain file or directory we want to block. At the beginning, I was saving the whole char * filename in my data structure, but then It took me a while to figure out that each file has a unique ID. The ID is set inside the inode, and it is called unsigned long i_ino. This variable holds the ID for every file or path/directory. This fileID will help my data structure to keep track of files by an ID instead of a char * filename.

Next the function int searchFileID() will be responsible of getting the file ID for each char* filename given. It will first check the length of the filename usng the function strnlen_user() which will be handy for me. I did create a function to give the length of a string but having that function made things a little easy. Later on the function I kmalloc the space for char* filename into the a “tempFile” kernel and I used the function copy_from_user to copy that char *filename from user to the new kernel space. I found this in the hard way when I was trying to get the filename from user space to kernel space. Next, I used the kernel_path() buildin function to find the path of the char *filename we just copied at the kernel. This is a very good function to find a path giving a filename. I got the struct path file and then getting the inode from the dentry of this file is going to be important because the dentry contains an inode and inside of this inode there is the variable that holds the ID for each file.

The next function is called int checkFileID(const struct path filePath) and it will be responsible to iterate the whole linked list that we have and if a file ID is found on our data structure it will return error -EINVAL. If there is not a match of a file ID then it will return 0 as normal. This function will put in the kernel file namei.c in the link_path_walk()function which are responsible make links for the absolute path.

The function called long fc421_reset(void) this function will be responsible to clear all the file ID on my data structure. It will check this has root level, and it will iterate through the list of “allBlockPorts” and it will erase them one by one. It is important to note that the write mutex will be present during this action

The function called long fc421_block_file(const char *filename) will be responsible to search the fileID, check that the file ID does not exit already in our list. If the fileID already exits it will return -EEXIST, if it doesn’t it will be created an added to our list. It will return 0 on success. It is important to mention that the write mutex will be present during the creation of this new blockFile struct.

The function called long fc421_unblock_file(const char *filename) will be responsible to search the fileID in our data structure. If during the iteration of my list, the file appear with the corresponding ID it will be removed from the list. If it does not match, it will return an error with – ENOENT. The user has to be root in order to do this because it is checking for root at the beginning. During this phase, the write mutex will be present to avoid any problem with the critical section.

The function called long fc421_query(const char *filename) will be responsible to iterate in my list and find a fileID with a given char * filename. In order to get the ID of the file I will be

calling the function searchFileID() which will give me the fileID. Next, once I have the fileID, I will be iterating in my list and matching the ID with fileID saved in my liked list. If a match is found, It will return the count of how many times certain fileID was tried to access. If the file was not found it will return error – ENOENT.

Lastly, the function in the kernel I will be modifying is the link_path_walk(). It took me a lot of time to figure out this function. I was looking at all the functions that use the path and I ended up with at least 4 functions that deal with the path for a file in one way or another. The reason this function is very important is because this function iterate through the char and finds the file avoiding "/" and NULLs. I will putting my function checkFileID() which it will be at the end once the dentry is found and I will pass it and deal with it in my function. In other words, it will find the fileID and match it with any in my data structure and if already exist then I will throw an error preventing to read/modify or write to this file/directory.

4. References

For small part where I had to do debugging, I have put links on sections on my code as comment about where I got the answer to solve some issues.

"Linux Source Code: Fs/Namei.c (v3.4)." *Bootlin*, elixir.bootlin.com/linux/v3.4/source/fs/namei.c#L1711.

"Linux Source Code: Net/Socket.c (v5.1)." *Bootlin*, elixir.bootlin.com/linux/latest/source/net/socket.c.

Sahu, Vishal SahuVishal. "Difference between mutex_init and DEFINE_MUTEX." *Stack Overflow*, stackoverflow.com/questions/33932991/difference-between-mutex-init-and-define-mutex.

DzangoDzango 111, and Alex HoppusAlex Hoppus 2. "How Does path_lookup in Linux Kernel Work?" *Stack Overflow*, stackoverflow.com/questions/31342628/how-does-path-lookup-in-linux-kernel-work.

Struct sockaddr_in, Struct in_addr, www.gta.ufrj.br/ensino/eel878/sockets/sockaddr_inman.html.

strlen_user, www.fsl.cs.sunysb.edu/kernel-api/re250.html.